컴포넌트 인터페이스의 행위 호환성을 검증하는 도구 설계

김호준, 이우진
경북대학교 전자전기컴퓨터학부
sisqo00@nate.com, woojin@knu.ac.kr

Design of Behavior Conformance Verifier for Component Interface

HoJun Kim, Woo Jin Lee
School of Electrical Engineering and Computer Science, Kyungpook National University

요 약

컴포넌트 기반 개발(Component Based Development, CBD)은 높은 품질의 소프트웨어를 신속하고 효과적으로 개발할 수 있는 방법으로써 각광 받아 왔다. 하지만 CBD 를 이용한 기존의 소프트웨어 개발은 컴포넌트의 행위를 배제하고 컴포넌트 인터페이스만 참조하여 컴포넌트를 설계함으로써, 컴포넌트의 구체적인 행위에 대한 파악과 컴포넌트 간 인터페이스 호환성 보장이 불가능하였다. 이에 따라 컴포넌트 설계 단계에서 컴포넌트의 행위를 상태머신으로 표현하고, 표현된 상태머신을 통해 컴포넌트의 행위 호환성을 보장할 필요가 있다. 이 연구에서는 상태머신으로 표현된 컴포넌트의 행위를 관찰 일치(observation equivalence)와 호출 일관성(invocation consistency)의 개념을 이용하여 행위 호환성을 검증하는 방법을 제공하고, 동적으로 이를 수행하는 도구를 설계한다.

1. 서론

소프트웨어 개발에 있어서, 컴포넌트를 기반으로 하여 소프트웨어의 재사용성을 제공해주는 소프트웨어 개발 방법론을 컴포넌트 기반 개발 방법론(Component Based Development, CBD) [1]이라 한다. 최근 이에 대한 관심이 고조되면서 RUP[2], Catalysis[3] 등과 같은 CBD 방법론이 소개되었고, Sun 의 J2EE 와 Microsoft 의 .NET 과 같은 CBD 플랫폼이 확산됨에 따라 더욱 주목 받고 있다.

일반적으로 CBD 를 이용한 기존의 소프트웨어 개발에서는 컴포넌트 행위에 대한 별다른 검증 없이 컴포넌트 인터페이스의 명세만을 참조하여 컴포넌트를 설계해왔다. 하지만 인터페이스의 명세만으로는 컴포넌트의 구체적인 행위에 대한 파악이 불가능하므로, 사용자 입장에서 컴포넌트의 구체적인 사용 패턴 파악이 불가능하다. 또한, 하나의 컴포넌트의 제공 인터페이스와 또 다른 컴포넌트의 요구 인터페이스가 인터페이스의 명세만을 참조하여 서로 연결될 시에는, 요구 인터페이스가 요구하는 행위를 제공 인터페이스가 올바로 제공한다는 보장이 없게 된다. 따라서 컴포넌트 인터페이스의 행위 호환성의 검증이 필요하며, 이를 컴포넌트의 설계 단계에서 수행하면 컴포넌트의 개발 비용적인 측면에서 더욱 향상을 기할 수 있다.

※ 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.(2008-SW-32-DM-05)

컴포넌트 호환성 검증은 정형적 방법(Formal Method)을 이용하여 추상적 행위에 대한 구체적 행위의 대체 가능성 검증에 대한 연구가 주로 이루어지고 있는데, 일반적으로 정형적 방법은 큰 규모의 시스템에 적용하기가 어렵고 복잡한 단점이 있다. 본 논문에서는 단순한 행위 대체성을 보완한 컴포넌트 인터페이스의 행위 호환성 검증과 이를 동적으로 수행하는 도구를 제안한다. 이를 위해 UML 컴포넌트[4]의 개념을 사용하여 컴포넌트 인터페이스의 행위를 상태머신(State Machine)으로 표현하고 이를 비교적 간단한 Labeled Transition System(LTS)[5]로 변형한 후, 이들 LTS 모델 간의 합성 모델 분석을 통하여 컴포넌트의 행위 호환성을 검증하는 방법을 제안한다. 논문의 구성으로는 제 2 절에서 컴포넌트 인터페이스의 행위 호환성 검증에 대해 기술하고, 제 3 절에서는 컴포넌트 인터페이스의 행위 호환성 검증 도구의 설계를 제안한다. 마지막으로 제 4 절에서는 결론과 향후 연구 방향을 제시한다.

2. 컴포넌트 인터페이스 행위 호환성 검증

앞 절에서 언급된 것처럼, 컴포넌트의 설계 단계에서 컴포넌트의 구현 및 조합 시에는 반드시 컴포넌트의 행위적 관점에서 컴포넌트 인터페이스의 호환성을 검증할 필요가 있다. 컴포넌트 인터페이스의 호환성 검증의 방향은 다음과 같은 두 가지 관점으로 생각해 볼 수 있다.

○ 컴포넌트 구현 검증
: 컴포넌트 행위의 명세를 인터페이스에 표현하고, 컴포넌트 내부의 구체적인 행위의 컴포넌트 인터페이스 행위 명세에 대한 만족을 검증한다. 이를 통해 인터페이스를 통한 컴포넌트의 행위 파악이 가능하며, 행위 명세에 대한 행위 구현의 보장이 이루어진다.

○ 컴포넌트 간 인터페이스 연결성 검증
: 컴포넌트 간 제공 인터페이스와 요구 인터페이스의 연결 시에 각각의 인터페이스에 대한 행위 명세를 표현하고, 두 행위의 호환성을 검증함으로써 요구 인터페이스의 요구 행위에 제공 인터페이스의 제공 행위가 만족됨을 보장할 수 있다.

기존의 CBD 기반 소프트웨어 개발에서 간과되었던 문제점을 위와 같은 두 가지 관점을 통한 컴포넌트 인터페이스의 행위 호환성 검증 문제로 생각해봄으로써 이를 해결할 수 있다.

UML 에서는 컴포넌트의 행위를 상태머신(state machine)[4]으로 표현하며, 본 연구에서는 상태머신을 비교적 간단한 Labeled Transition System(LTS)로 변형하여 컴포넌트 인터페이스의 호환성을 검증한다. 이를 위해 먼저 LTS 와 필요한 심볼들에 대해 정의한다.

정의 1. ***Labeled Transition System(LTS)***

LTS = (S, L*, { ⇒s : s ∈ L* })

- S : 상태들의 집합
- L* : 관찰 가능한 행위(L)들의 시퀀스
- ⇒s : 모든 행위들의 시퀀스로부터 발생 가능한 천이 관계로 t = $\alpha_1 ... \alpha_n$ ∈ Act* 일 때, E(→τ)* →α_1 (→τ)* ... (→τ)* →α_n (→τ)* E′ 이면 E ⇒t E′ 이다.

위의 정의에서 "τ"는 관찰 가능하지 않은 행위, "Act"는 모든 행위를 뜻한다. 즉, Act = L ∪ {τ}이다. 따라서, 이들 정의를 통해 컴포넌트 행위를 상태머신으로 표현하고 이를 다시 LTS 로 표현한다. 이를 이용하여 앞 절의 두 가지 관점에 대한 컴포넌트 인터페이스의 호환성 검증 방법을 다음과 같이 제시한다.

2.1 컴포넌트 구현 검증 방법

컴포넌트 구현에 대한 검증은, 컴포넌트 행위의 명세에 해당하는 인터페이스의 프로토콜 상태머신(protocol state machine)[4]과 컴포넌트 내부의 구체적인 행위에 해당하는 컴포넌트의 상태머신 간에 이루어진다. 두 상태머신은 추상화 정도가 다를 수 있으며 구현이 명세를 만족하기 위해서는 추상화 정도가 동일한 관찰 가능한 행위의 일치 검사가 이루어야 한다. 즉, 명세에 대한 행위가 구현에서 만족되어야 하며, 명세 되지 않은 행위는 구현에 포함되지 않아야 함을 뜻한다. 이는 관찰 일치(observation equivalence)를 뜻하며, 이를 weak bisimulation 기법[5]을 적용하여 수행한다.

정의 2. ***Weak Bisimulation***

두 LTS P, Q 의 2 원 관계 S ⊆ P x Q 에서 모든 α ∈ Act 에 대해 다음을 만족하면 weak bisimulation 이다. 여기에서 t^ ∈ L* 는 t ∈ Act* 에서 모든 τ 를 제거한 시퀀스이다.

i) 모든 P →α P′ 에 대하여 Q ⇒α^ Q′ 이고 (P′ ,Q′) ∈ S 인 Q′가 존재한다.

ii) 모든 Q →α Q′ 에 대하여 P ⇒α^ P′ 이고 (P′ ,Q′) ∈ S 인 P′가 존재한다.

2.2 컴포넌트 간 인터페이스 연결성 검증 방법

컴포넌트 간 인터페이스 연결성 검증은 요구 인터페이스와 제공 인터페이스의 두 프로토콜 상태머신 간에 이루어지는데, 이 역시 두 머신의 추상화 정도가 다를 수 있으며 추상화 정도가 높은 요구 인터페이스의 행위가 제공 인터페이스에서 만족되어야 한다. 이는 호출 일관성(invocation consistency)[6]을 뜻하며, 다음과 같이 정의한다.

정의 3. 호출 일관성*(invocation consistency)*

두 LTS P, Q 의 2 원 관계 S ⊆ P x Q 에서 모든 α ∈ Act 에 대해 다음을 만족하면 P 에 대해 Q 가 호출 일관성을 갖는다.

- 모든 P →α P′ 에 대하여 Q ⇒α Q′ 이고 (P′ ,Q′) ∈ S 인 Q′가 존재한다.

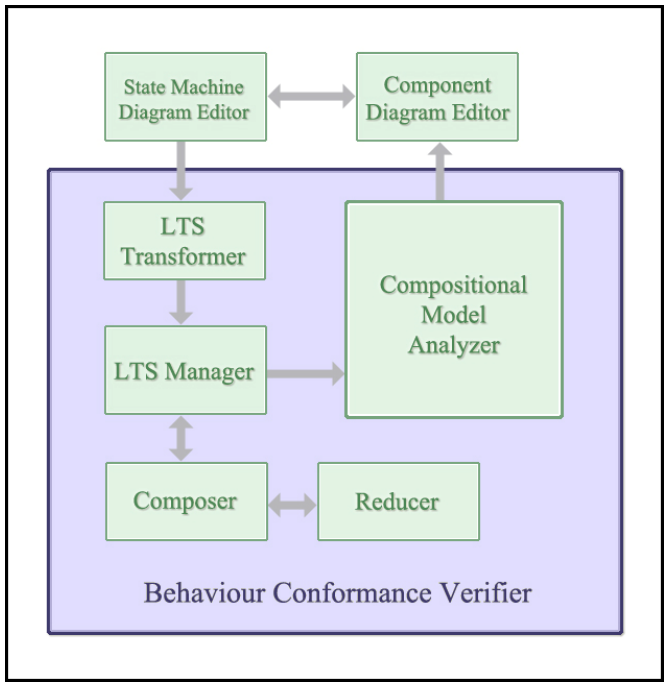
추상화 정도가 높은 요구 인터페이스의 행위를 LTS 모델 P 로 표현하고 추상화 정도가 낮은 제공 인터페이스의 행위를 LTS 모델 Q 로 표현하여 위에 정의된 호출일관성으로 검증하면 컴포넌트 간 인터페이스의 연결성에 관한 검증을 수행할 수 있다.

두 가지 행위 호환성 검증 모두 추상화 정도가 다른 두 LTS 모델간의 행위를 두 LTS 모델을 합성하여 분석한다. 이 때, 추상화 정도가 높은 행위에서 보이지 않는 전이, 즉, 추상화 정도가 낮은 행위에서만 나타나는 전이를 "τ"로 생각하고 두 행위 간의 관찰 일치나 호출 일관성을 분석함으로써 컴포넌트의 행위 호환성을 검증할 수 있다.

3. 컴포넌트 인터페이스 행위 호환성 검증 도구 설계

컴포넌트의 행위 호환성 검증을 수행하기 위해서 먼저 컴포넌트를 설계할 수 있는 컴포넌트 다이어그램 편집기와 컴포넌트의 행위를 기술할 수 있는 상태

머신 다이어그램 편집기를 제안한다. 상태 머신 다이어그램 편집기와 연계되어 상태머신을 간단한 LTS 모델로 변형시켜 주는 LTS 변환기와 변환된 LTS 모델을 기술하는 LTS 매니저가 필요하다. 상태머신은 영역(region)[4]을 가지며, 하나의 상태머신은 가지고 있는 영역의 수만큼의 LTS 모델로 변형된다. 이를 하나의 LTS 모델로 표현하기 위해 여러 LTS 모델을 합성해야 하는데, 합성 시에 축약과 합성을 반복함으로써 상태 폭발(state explosion) [7]을 방지하고 합성된 모델을 간소화할 수 있다. 이를 위해 여러 LTS 모델을 하나의 합성된 LTS 모델로 만들어주는 감축기와 합성기가 필요하다. 이렇게 하나의 LTS 모델로 표현된 각 상태머신을 이용하여 궁극적으로 컴포넌트의 행위 호환성을 수행하는 합성 모델 분석기를 제안한다. 이들 도구들을 바탕으로 컴포넌트 다이어그램 편집기는 합성 모델 분석기의 검증 보고기와 연계하여 동적으로 행위 호환성을 검증 결과를 볼 수 있게 한다. 그림 1 은 이들 도구의 구성도를 나타낸다.

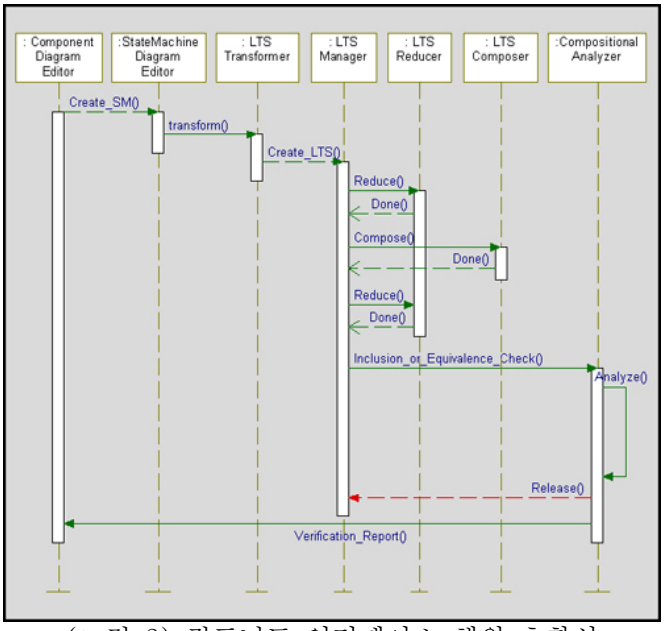


(그림 1) 컴포넌트 인터페이스 행위 호환성 검증 도구의 구조

따라서 컴포넌트 인터페이스 행위 호환성 검증은 두 가지 다이어그램 편집기와 행위 호환성 검증기가 서로 연계하여 이루어지며, 사용자 입장에서는 컴포넌트를 설계하는 과정에서 호환성 검증을 요청하고 검증 결과 보고를 확인하는 과정을 컴포넌트 다이어그램 편집기 상에서 수행하게 된다. 컴포넌트 인터페이스 호환성 검증 과정의 전체적인 수행 과정에서 도구 내의 각 컴포넌트 간 상호작용 매커니즘은 그림 2 와 같이 나타낼 수 있다.

4. 결론 및 향후 연구 방향

소프트웨어 개발을 위한 방법론으로 소프트웨어의 재사용성을 제공하는 컴포넌트 기반 개발 방법론(Component Based Development, CDB)[1]이 최근 각광받고 있다. 본 논문에서는 기존의 CBD 를 이용한 소프트웨어 개발에서의 문제점을 해결하기 위해 컴포넌트 인터페이스 호환성 검증 방법을 제시하고 이를 동적으로 수행하는 도구를 설계하였다. 컴포넌트의 설계 및 구현, 조합 과정에서 동적으로 컴포넌트 인터페이스 호환성 검증을 수행하는 도구를 이용함으로써 의미적으로 명확한 설계가 컴포넌트 설계 단계에서 가능해짐으로써 컴포넌트 재사용성 향상을 기대할 수 있다. 설계된 도구는 Eclipse 에서 플러그-인 형태로 제공하는 GMF(Graphical Modeling Framework) [8]를 기반으로 구현에 대해 연구할 예정이다.



(그림 2) 컴포넌트 인터페이스 행위 호환성 검증 시나리오

참고문헌

[1] Keith Short, “Component-Based Development and Object Modeling,” Sterling Software, 1997
[2] Rational Software Corporation, “Rational Unified Process,” A Rational Software Corporation White Paper, 1998
[3] Desmond F. D’Souza, Alan Cameron Willis, Objects, Components, and Frameworks with UML : The Catalysis Approach, Addison Wesley, 1998
[4] OMG, “Unified Modeling Language : Superstructure,” version 2.1.1, 2007
[5] Robin Milner, Communication and Concurrency, Prentice Hall, 1989
[6] Martin Hitz, Gerty Kappel, “Developing with UML – Some Pitfalls and Workarounds,” 1999
[7] Inhye Kang, Insup Lee, Young-Si Kim, “An Efficient State Space Generation for the Analysis of Real-Time Systems,” IEEE Transaction on Software Engineering, Vol. 26, No.5, 2000
[8] Graphical Modeling Framework (GMF) version 2.0.2, 2008. Available at http://download.eclipse.org/modeling/gmf/downloads/index.php